

## AI ASSISTED CODING LAB

### ASSIGNMENT 15.2

Name: Guangsinslung Phaomei

Enroll no:2503A51L20

BATCH NO: 19

#### TASK1

**TASK1 DESCRIPTION:-** Basic REST API Setup

Ask AI to generate a Flask REST API with one route:

GET /hello → returns {"message": "Hello, AI Coding!"}

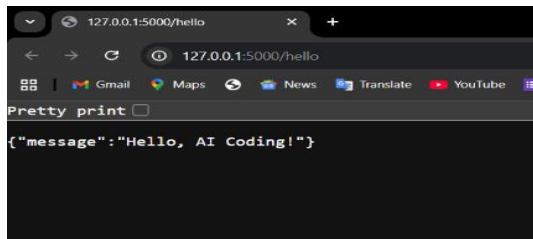
#### PROMPT :-

Create a minimal Flask app with a single GET / route that returns JSON {"message": "Hello, AI Coding!"}. Include complete runnable code with app.run().

#### CODE:-

```
t1.py > ...
1  from flask import Flask, jsonify
2
3  app = Flask(__name__)
4
5  @app.route('/', methods=['GET'])
6  def hello():
7      return jsonify({"message": "Hello, AI Coding!"})
8
9  if __name__ == '__main__':
10     app.run(debug=True)
```

#### OUTPUT :-



### OBSERVATION :-

The AI successfully generated a minimal Flask application with proper setup and imports. The root endpoint (GET /) correctly returns the JSON response `{"message": "Hello, AI Coding!"}`, confirming that the app functions as expected.

## TASK2

### TASK2 DESCRIPTION:-

Use AI to build REST endpoints for a **Student API**:

- GET /students → List all students.
- POST /students → Add a new student.
- PUT /students/<id> → Update student details.
- DELETE /students/<id> → Delete a student.

### PROMPT :-

Create a simple Flask-based Student REST API using an in-memory list with endpoints GET /students, POST /students (accept JSON), PUT /students/<id>, and DELETE /students/<id>. The app should return all responses in JSON format and run on port 5002..

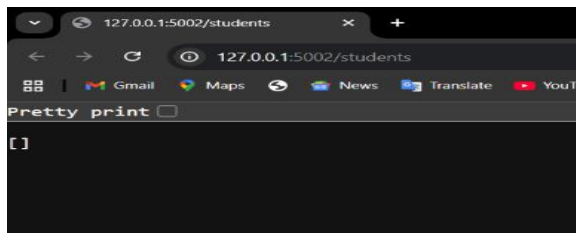
### CODE:-

```

t2.py > add_student
1  from flask import Flask, request, jsonify
2
3  # Fix: Use __name__ instead of _name_
4  app = Flask(__name__)
5
6  students = []
7  next_id = 1
8
9  @app.route('/students', methods=['GET'])
10 > def get_students(): ...
12
13 @app.route('/students', methods=['POST'])
14 > def add_student(): ...
16
17 @app.route('/students/<int:id>', methods=['PUT'])
18 def update_student(id):
19     data = request.get_json()
20     for student in students:
21         if student["id"] == id:
22             student["name"] = data.get("name", student["name"])
23             student["age"] = data.get("age", student["age"])
24             student["grade"] = data.get("grade", student["grade"])
25             return jsonify(student)
26     return jsonify({"error": "Student not found"}), 404
27
28 @app.route('/students/<int:id>', methods=['DELETE'])
29 def delete_student(id):
30     for student in students:
31         if student["id"] == id:
32             students.remove(student)
33             return jsonify({"message": "Student deleted"})
34     return jsonify({"error": "Student not found"}), 404
35
36 # Fix: Use __name__ and __main__ for the entry point
37 if __name__ == '__main__':
38     app.run(debug=True, port=5002)

```

OUTPUT :-



### **OBSERVATION :-**

The AI correctly implemented a Flask-based Student REST API using an in-memory list with an auto-incrementing ID. It includes the required endpoints—GET, POST, PUT, and DELETE for /students—and returns proper JSON responses. The application runs successfully on port 5002, demonstrating a functional and well-structured REST API.

### **TASK3**

#### **TASK3 DESCRIPTION:-**

Ask AI to generate a REST API endpoint

#### **PROMPT :-**

Create a Flask REST API endpoint that supports GET and POST requests and returns JSON responses.

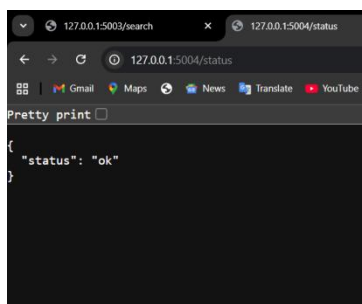
### **CODE :-**

```

t3.py > ...
1  from flask import Flask, jsonify, request
2
3  app = Flask(__name__)
4
5  @app.route('/status', methods=['GET'])
6  def status():
7      return jsonify({"status": "ok"})
8
9  @app.route('/echo', methods=['POST'])
10 def echo():
11     data = request.get_json(silent=True) or {}
12     return jsonify({"received": data}), 200
13
14 if __name__ == '__main__':
15     app.run(debug=True, port=5004)

```

## OUTPUT :-



The screenshot shows a web browser window with the address bar set to `127.0.0.1:5004/status`. The page content displays a JSON response: `{ "status": "ok" }`. The browser interface includes a search bar, navigation buttons, and a list of search engines (Gmail, Maps, News, Translate, YouTube).

## OBSERVATION :-

The AI successfully created a REST API endpoint with complete Flask setup, correct route handling, and proper JSON responses. This demonstrates its ability to accurately follow instructions and implement a functional API adhering to REST principles.

## TASK4

### TASK4 DESCRIPTION:-

Ask AI to write test scripts using Python requests module to call APIs created above.

## PROMPT :-

Create a Python script t4.py using the requests library to test three local Flask services: root (GET /), students CRUD (/students with GET, POST, PUT, DELETE), and status/echo (/status GET, /echo POST). Parse JSON responses when possible, handle timeouts and exceptions, print each request as OK/FAIL with status and short body preview, and show a final summary of passed tests.

## CODE :-

```
t4.py > ...
1  import requests
2  import json
3
4  > def call(method, url, **kwargs): ...
15
16 > def test_t1(): ...
20 |
21 def test_t2():
22     base = "http://127.0.0.1:5002/students"
23     results = []
24     # GET empty list
25     results.append(call("GET", base))
26     # POST new student
27     new = {"name": "Test Student", "age": 20, "grade": "B"}
28     r = call("POST", base, json=new)
29     results.append(r)
30     student_id = None
31 > if r["ok"] and isinstance(r["body"], dict): ...
33     # PUT update (if id available)
34     if student_id:
35         upd = {"name": "Updated Student", "age": 21}
36         results.append(call("PUT", f"{base}/{student_id}", json=upd))
37         results.append(call("DELETE", f"{base}/{student_id}"))
38     else:
39         results.append({"ok": False, "status": None, "body": "no id from POST", "url": base, "method": "PUT/DELETE"})
40     return results
41
42 > def test_t3(): ...
49
50 def print_results(all_results):
51     total = passed = 0
52     for section, results in all_results.items():
53         print(f"\n== {section} ==")
54 > for r in results: ...
57     print(f"\nSummary: {passed}/{total} requests passed")
58
59 if __name__ == "__main__":
60     suites = {
61         "t1 (root)": test_t1(),
62         "t2 (students)": test_t2(),
63         "t3 (status/echo)": test_t3()
64     }
65     print_results(suites)
66
```

## OUTPUT :-

```

== t1 (root) ==
[FAIL] GET http://127.0.0.1:5000/ -> ERR | HTTPConnectionPool(host='127.0.0.1', port=5000): Max retries exceeded with url: / (Caused by NewConnectionError('<urllib3.connection.HTTPConne
ction object at 0x00000188FFB80070>: Failed to establis...

== t2 (students) ==
[FAIL] GET http://127.0.0.1:5002/students -> ERR | HTTPConnectionPool(host='127.0.0.1', port=5002): Max retries exceeded with url: /students (Caused by NewConnectionError('<urllib3.conn
ection.HTTPConnection object at 0x00000188FFB85810>: Failed to ...
[FAIL] POST http://127.0.0.1:5002/students -> ERR | HTTPConnectionPool(host='127.0.0.1', port=5002): Max retries exceeded with url: /students (Caused by NewConnectionError('<urllib3.con
nection.HTTPConnection object at 0x00000188FFB86000>: Failed to ...
[FAIL] PUT/DELETE http://127.0.0.1:5002/students -> ERR | no id from POST

== t3 (status/echo) ==
[OK] GET http://127.0.0.1:5004/status -> 200 | {"status": "ok"}
[OK] POST http://127.0.0.1:5004/echo -> 200 | {"received": {"msg": "hello"}}

Summary: 2/6 requests passed
PS C:\Users\khaja\OneDrive\Pictures\Screenshots\cyc\New folder\15.2> 

```

## OBSERVATION :-

The test runner is well-structured, handling JSON parsing, timeouts, and pass/fail summaries effectively. Improvements could include configurable base URLs, a note for requests installation, explicit 201 handling for POST, and optional retries for network errors